# AMBER experiment's online filter system for virtualised IT infrastructure

V. Frolov, S. Huber, V. Jarý, I. Konorov, J. Nový, D. Ecker, B. M. Veit, M. Virius, **M. Zemko**

24th IEEE Real Time Conference

22nd April 2024

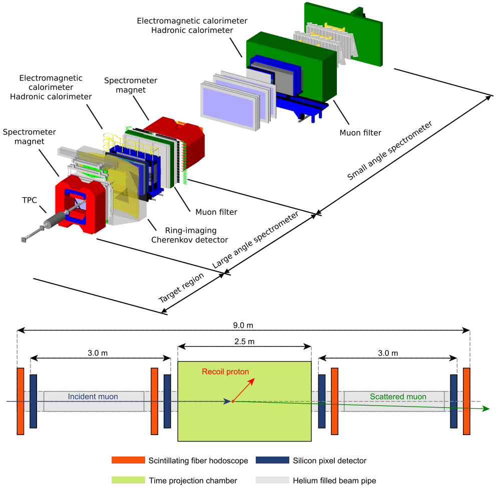
Apparatus for Meson and Baryon Experimental Research

FACULTY OF NUCLEAR SCIENCES AND PHYSICAL ENGINEERING CTU IN PRAGUE

FACULTY OF MATHEMATICS AND PHYSICS Charles University

# Outline

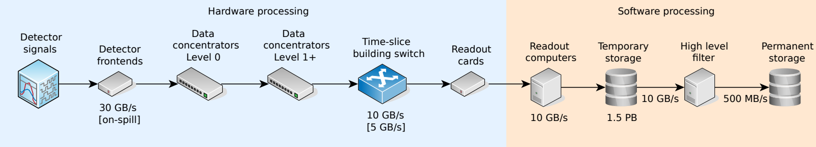
# AMBER experiment




- AMBER is a fixed target experiment located at the M2 beam line of the CERN SPS
- It has been approved by the CERN research board in 2021
- Measurement of the proton radius on an active hydrogen time projection chamber (TPC) with a muon beam is one of the objectives of the experiment
- Slow detectors (such as hydrogen TPC) have very long drift time (approx. 120 μs)
- They can handle only low trigger rates → need for a novel triggering approach

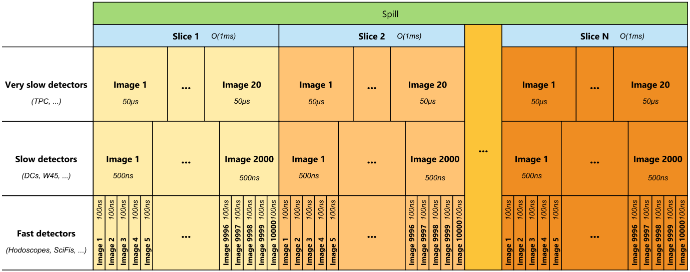
# Streaming readout system

- AMBER will use a trigerless data acquisition system based on continuous readout of detectors
- First stage relies on hardware-based processing in FPGA cards
- Second stage utilizes software developed with the Qt framework
- High-level filter is used instead of a low-level trigger logic
- General reduction scheme → any detector can participate in the filter decision



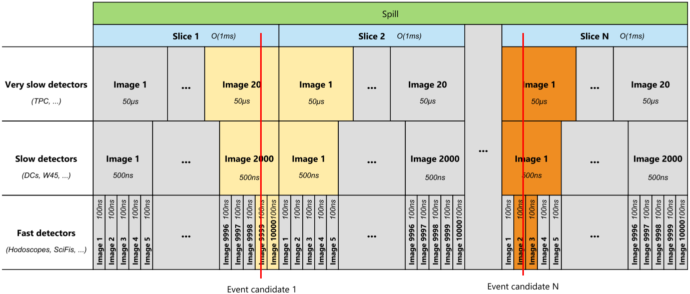
# Data structure

- We developed a custom streaming protocol consisting of several layers
- Data format consist of so-called time slices and images
- Global time slice signal is distributed to all detectors
- Images are generated individually based the detector time resolution
- Data reduction involves removing images that do not contain any physics events
- We store two consecutive images for every event candidate to prevent edge cases
- We expect a significant data reduction, aiming for a reduction factor of 100x



# Data structure

- We developed a custom streaming protocol consisting of several layers
- Data format consist of so-called time slices and images
- Global time slice signal is distributed to all detectors
- Images are generated individually based the detector time resolution
- Data reduction involves removing images that do not contain any physics events
- We store two consecutive images for every event candidate to prevent edge cases
- We expect a significant data reduction, aiming for a reduction factor of 100x

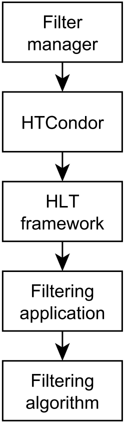


# High-level filter

- High-level filter is a distributed software platform for large-scale semi-online data filtering
- Filter runs on the virtualised CERN infrastructure shared with other users → significant cost optimization
- It consists of 2 main components:
  1. **Filter management system** (production system)
     - handles and manages individual requests
     - spawns instances of the filtering application
  2. **Filtering framework**
     - performs actual data analysis and reduction
     - includes additional tools for data browsing, quality monitoring

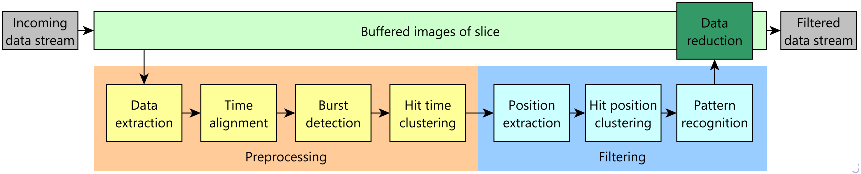
# Filtering framework

- High-throughput computational and reconstruction software written in optimized C++ and Qt
- It includes numerous libraries for various tasks such as data manipulation, detector alignment, database access, and more
- Algorithms vary depending on the current physics programme → modular architecture with many combinations
- Application optimizes the total performance through various methods:
  - Message-based thread communication
  - Non-uniform memory access (NUMA)
  - Adaptive multithreading
  - Zero-copy approach



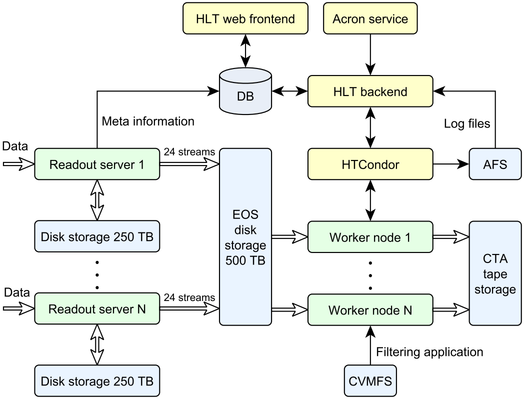
# Filtering pipeline

- We want to preserve and store only "interesting events"
- Such events can have different signatures, e.g., trajectories, coincidences, energy levels → track reconstruction is needed
- Initially, images are sorted into two exclusive groups:
  - **Primary images** – produced by detectors participating in filter decision
  - **Secondary images** – other images to be filtered
- We analyse primary images to make a decision using two phases:
  - **Time alignment** – processing and calibration of timestamps
  - **Spatial analysis** – analysis of hit positions to determine particle trajectory



# Filter management system

- Main task of management system is to handle filtering requests and follow them through the process
- It consists of two main parts:
  - **Backend** – based on HTCondor scheduling platform
  - **Frontend** – end user web application, serving as the main user interface
- Filtering system runs in the private CERN cloud shared with other experiments

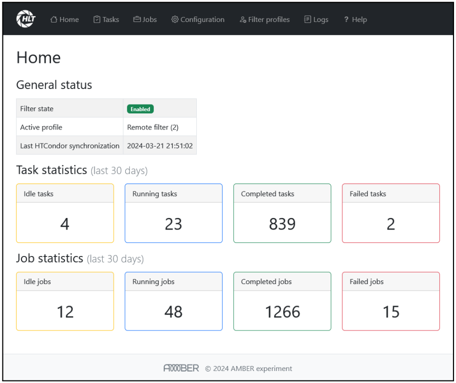


# Filter backend

- Python application is responsible for managing and submitting filtering jobs
- It relies on the HTCondor system as the primary execution platform
- HTCondor efficiently distributes computing tasks to connected machines → ensures scalability of our filter system
- CERN HTCondor instance hosts more than 100,000 CPU cores
- Jobs are synchronized through their lifecycle with HTCondor states
- Backend checks for any incoming requests every 2 minutes and submits filtering jobs to HTCondor

HTCondor Software Suite

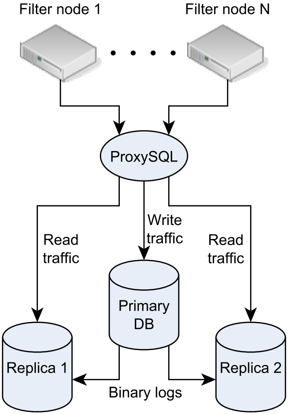
# Filter frontend



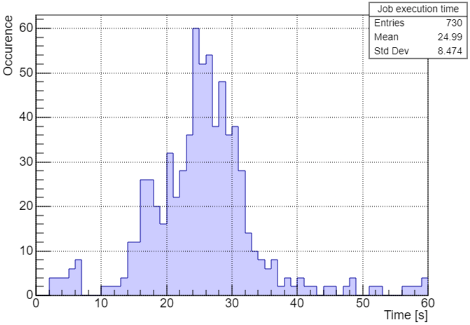

- Frontend is a web-based application for user interaction with the filter
- Users can control, monitor, and submits filtering tasks with custom parameters
- It provides monitoring and overview to operators on shift
- Tasks are automatically generated with predefined settings by default
- Filter also supports reprocessing of data files with different settings

# Configuration and database access

- During the filter operation, thousands instances of filtering application connect to the database simultaneously → we implemented a clustered database design
- It consists of a single primary database, two replicas, and ProxySQL load balancer
- MySQL caching mechanism quickly responds to any repetitive queries from nodes
- Database operates within a container on the Database-on-demand service
- Daily backups and recovery procedures are efficiently managed by the DBOD service



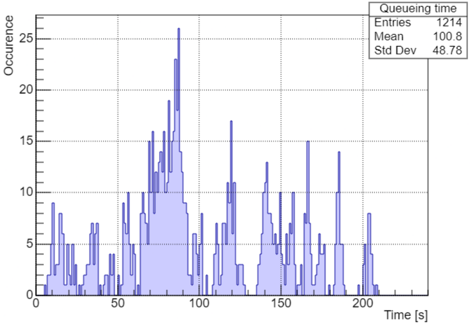
# Performance benchmark



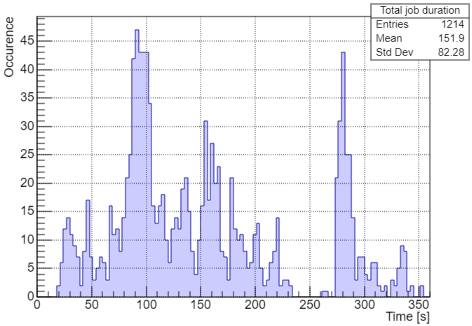

- We performed several tests and latency analysis of the filtering system
- Each file was processed by a single job running on 4 CPU cores
- Filtering a data file took **24.99 ± 8.47 seconds** on average
- Average processing rate is **10.24 MB/s**
- Considering 10 GB/s nominal data rate, we need roughly 1,000 CPU cores
- Processing rate depends on:
    - CPU performance,
    - used filter algorithm,
    - number of projections,
    - slice duration, etc.

# HTCondor queueing time

- Second measurement examined the queueing time in HTCondor
- Our initial tests indicates that a job spends around **100.80 ± 48.78 seconds** in the queue
- Variability is significant, almost 50 %
- Despite this volatility, the range of queueing times we observed is still within acceptable limits for our needs
- Queueing time is affected by:
  - cluster occupancy (other jobs),
  - job requirements (lower is better),
  - user priority (higher is better),
  - HTCondor quotas, etc.
- If queue times were to increase dramatically, we have several response strategies to address it



# Total job duration




- Third measurement focused on the total time taken for jobs to complete
- This includes factors like delays from the backend, file transfers, initialization processes, and so on
- On average, we found that jobs took about **151.90 ± 82.28 seconds** to complete
- Due to the limited statistical data, these measured values are preliminary
- Backend contributed an additional minute of latency on average due to 2 minute synchronization period

# Summary

- We designed the streaming acquisition system for the AMBER experiment at CERN
- System includes the custom data protocol and the high-level filtering framework replacing the low-level trigger
- We developed a scalable high-throughput filtering management system running on virtualised CERN infrastructure based on HTCondor, EOS, DBOD, CVMFS, and other services
- Performance of the filtering system has been measured and optimized
- System is capable of processing at least **10 GB/s** data rate in a semi-online manner with an average latency of **151.90 seconds**
- Filter will be used in the upcoming proton radius measurement later this year and will be further tested

# Thank you for your attention